CSC161 2011S *Imperative Problem Solving*

Laboratory: Testing

Summary:

Prerequisites: Familiarity with functions, separate compilation, and arrays.

- Preparation
- Exercises
  - Exercise 1: Simple Tests
  - Exercise 2: An Interactive Tester
  - Exercise 3: Systematic Testing
- For Those With Extra Time

Preparation

a. Create a directory for this lab.

b. Copy into that directory two files:

- `/home/rebelsky/share/sorting/bs.h`
- `/home/rebelsky/share/sorting/jqbs.c`

c. Read `bs.h` to familiarize yourself with the purpose and form of the `binary_search` routine.

Exercises

Exercise 1: Simple Tests

Let's start with some very simple tests for the routine.

a. Create a file, `bstest1.c`, that contains a `main` routine that implements the following test:

- For the array `{ 1, 5, 9, 21 }`, `binary_search` returns 0 when looking for 1.

b. Create a Makefile to build a tester. You might use a rule like

```
jqtest1: bstest1.o jqbs.o
	$(CC) -o jqtest1 bstest1.o jqbs.o
```

c. Build and run your tester. Did you find any errors?

d. Add some more tests.

- For the same array, `binary_search` returns 1 when looking for 5.
- For the same array, `binary_search` returns 3 when looking for 21.
- For the same array, `binary_search` returns -1 when looking for 0.
- For the same array, `binary_search` returns -1 when looking for 7.
- For the same array, `binary_search` returns -1 when looking for 22.
- Any other tests you deem appropriate.

e. Build and run your tester. Did you find any errors?

Exercise 2: An Interactive Tester

a. Write a program, `bstest2.c`, that permits more interactive testing of `binary_search`. Your program should

- take the array to search from the command line
- repeatedly prompt the user for a number and report on the idnex of that number

Note: You may find it helpful to look at the code we wrote for the interactive tester for the `average` procedure.

b. Add an entry to the Makefile to build a tester for `jqbs.c`.

c. Build and run your tester. Did you find any errors?

Exercise 3: Systematic Testing

The tests in exercise 1 are not very systematic. For example, we never check what happens if the value is not in the array, and is between element 0 and element 1. (Surprisingly, there are many implementations of binary search which work correctly when looking between elements 1 and 2, but not between elements 0 and 1.) One advantage of using computers to do testing is that they can systematically generate a large number of tests. For example, if we generate the array { 0, 2, 4, 6, 8, 10 }, we know that

- For even numbers, *n*, between 0 and 10, *n* has index *n*/2.
- For odd numbers, *n*, between -1 and 11, *n* has index -1.

Write a tester that uses this strategy. You may also add any variants you consider appropriate.

Did you find any errors?

For Those With Extra Time

If you find yourself with extra time, implement binary search yourself.

Copyright © 2010 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.